

## 9 ПРОГРАМУВАННЯ РЕКУРСІЇ

**Мета:** Здобути практичні навички зі створення та використання рекурсивних функцій

### 9.1 Короткі теоретичні відомості

Означення називається *рекурсивним*, якщо воно задає елементи певної множини за допомогою інших елементів цієї самої множини. Об'єкт, заданий рекурсивним означенням, також називається рекурсивним, а використання таких означень – *рекурсією*.

Функції, що містять виклики самих себе, як і самі ці виклики, називаються *рекурсивними*.

Виклик рекурсивної функції виконується так само, як і виклик будь-якої функції. У рекурсивній функції обов'язково має бути умова, за істинності якої відбувається повернення з виклику. Ця умова визначає дно рекурсії, яке під час виконання функції обов'язково має досягатися, інакше виклики призведуть до переповнення програмного стека або інших непередбачуваних наслідків.

*Прямою* (безпосередньою) рекурсією називається рекурсія, при якій всередині тіла деякої функції міститься виклик тієї ж функції.

```
void fn(int i)
{   /* ... */
    fn(i);
    /* ... */
}
```

*Непрямою* рекурсією називається рекурсія, що здійснює рекурсивний виклик функції шляхом ланцюга викликів інших функцій (рис. 9.1). При цьому всі функції ланцюга, що здійснюють рекурсію, вважаються також рекурсивними.



Рисунок 9.1 — Непряма рекурсія

```
void fnA(int i);
void fnB(int i);
void fnC(int i);
```

```

void fnA(int i)
{
    /* ... */
    fnB(i);
    /* ... */
}
void fnB(int i)
{
    /* ... */
    fnC(i);
    /* ... */
}
void fnC(int i)
{
    /* ... */
    fnA(i);
    /* ... */
}

```

Якщо функція викликає сама себе, то в стеку створюється копія значень її параметрів, як і при виклику звичайної функції, після чого управління передається першому оператору функції. При повторному виклику цей процес повторюється.

З рекурсивними функціями пов'язано два важливих поняття – *глибина рекурсії* й *загальна кількість викликів*, породжених викликом рекурсивної підпрограми. Будемо відрізняти глибину рекурсії, на якій перебуває виклик, і глибину рекурсії, породжену викликом.

Глибина рекурсії, на якій перебуває виклик підпрограми – це кількість рекурсивних викликів, розпочатих і не закінчених у момент початку цього виклику.

Глибина рекурсії, породжена викликом – це максимальна кількість рекурсивних викликів, які розпочато й не закінчено після початку цього виклику.

Кожен екземпляр займає ділянку певного розміру, тому збільшення глибини, як було сказано вище, може призвести до переповнення програмного стека.

Загальна кількість рекурсивних викликів, породжених викликом рекурсивної функції, – це кількість викликів, виконаних між його початком і

завершенням. Загальна кількість породжених викликів визначає тривалість виконання виклику.

Рекурсивні функції найчастіше застосовують для компактної реалізації рекурсивних алгоритмів, а також для роботи зі структурами даних, описаними рекурсивно, наприклад з бінарними деревами. Кожну рекурсивну функцію можна реалізувати без застосування рекурсії. Достоїнством рекурсії є компактний запис, а недоліками – витрата часу й пам'яті на повторні виклики функції й передавання їй копій параметрів і, головне, небезпека переповнення стека.

## **9.2 Завдання**

Завдання 1. Створити функцію знаходження найбільшого спільного дільника (НСД) двох цілих чисел методом Евкліда (шляхом віднімання та ділення) з використанням рекурсії. Протестувати роботу програми. Зробити відповідні висновки.

### **9.3 Хід роботи**

#### **9.3.1 Постановка задачі**

*Дано:*  $n, m \in \mathbb{N}$ ;

*Визначити:*  $nsd \in \mathbb{N}$  - НСД чисел  $n$  та  $m$ .

#### **9.3.2 Метод реалізації інформаційного процесу**

НСД ( $m, n$ ) - це найбільше з чисел, на яке діляться і  $m$ , і  $n$ .

Суть алгоритму Евкліда – два числа порівнюють і від більшого віднімають менше до тих пір, поки числа не стануть рівними. Число, якому вони стануть рівними, і є їх найбільший спільний дільник.

(9.1)

#### **9.3.3 Алгоритм реалізації інформаційного процесу**

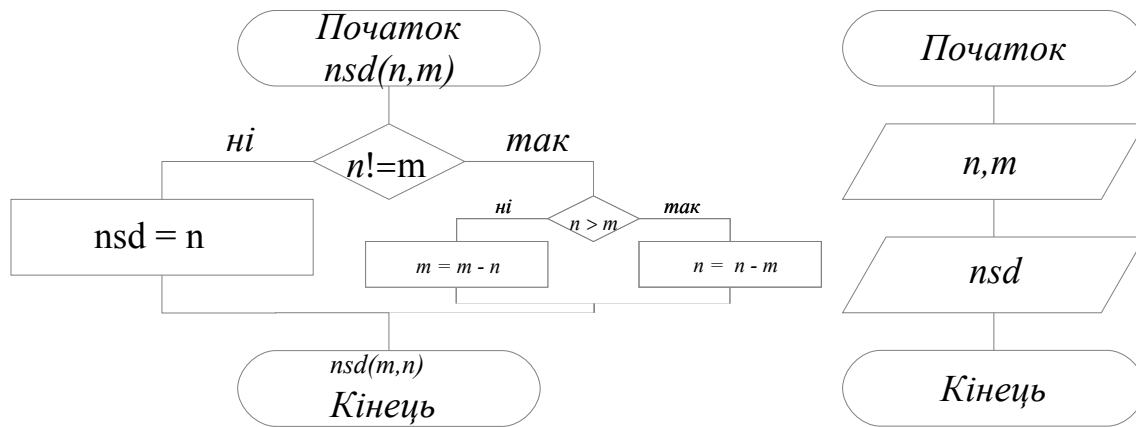


Рисунок 9.2 — Алгоритм розв'язку задачі

### 9.3.4 Програмування

Побудова таблиці ідентифікаторів.

Таблиця 9.1 — Таблиця ідентифікаторів

№ з/п	Змінна або константа	Ідентифікатор
1	$n$	n
2	$m$	m
3	$НСД$	nsd

Введення тексту програми:

```

#include <iostream>
using namespace std;

int nsd (int n, int m) {
    if (n != m) {
        if (n > m) n = n - m; else m = m - n;
    } else return n;
    return nsd(n, m);
}

int main() {
    int n, m;
    cout << "Input n="; cin >> n;
    cout << "Input m="; cin >> m;

    cout << nsd(n, m) << endl;
    system("pause");
    return 0;
}

```

### 9.3.5 Тестування та виявлення помилок

Для виявлення алгоритмічних помилок та вирішення проблеми достовірності отриманих результатів можна виконати обчислення у електронній таблиці і порівняти отримані розв'язки.

Для цього у електронній книзі “Обчислення функцій” Лист9 перейменовуємо на ЛР9 та виконуємо обчислення за формою:

n	m	НСД
18	45	=GCD(A2; B2)

Рисунок 9.3 — Розв'язок задачі у ЕТ

### 9.3.6 Обчислення, обробка і аналіз результатів

У ході виконання даної роботи отримано наступні результати:

```
Input n=18
Input m=45
9
sh: pause: command not found
Program ended with exit code: 0
```

Рисунок 9.4 — Результат обчислень

n	m	НСД
18	45	9

Рисунок 9.5 — Результат обчислень у електронній таблиці

Порівнюючи результати, отримані трьома різними способами з високою вірогідністю можна стверджувати, що обчислення виконано правильно, так як отримані значення співпали.

## 9.2 Завдання

Завдання 3. Написати рекурсивну функцію, яка обчислює мінімальний елемент масиву.

## 9.3 Хід роботи

### 9.3.1 Постановка задачі

Дано:  $a$ ,  $max \in \mathbb{N}$ ;

Визначити:  $min \in \mathbb{N}$  - мінімальний елемент масиву.

### 9.3.2 Метод реалізації інформаційного процесу

$min$  - мінімум усі елементи масиву.

### 9.3.4 Програмування

Побудова таблиці ідентифікаторів.

Таблиця 9.1 — Таблиця ідентифікаторів

№ з/п	Змінна або константа	Ідентифікатор
1	$a[]$	$a[]$
2	$max$	$max$
3	$arrayMin$	$arrayMin$

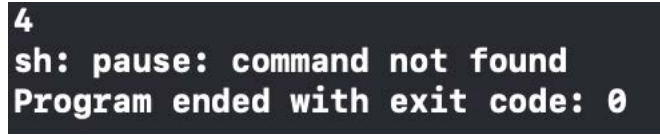
Введення тексту програми:

```
#include <iostream>
using namespace std;

int arrayMin (int array[], int i, int min) {
    if (i < 5) {
        if (min > array[i]) min = array[i];
    } else return min;
    i++;
    return arrayMin(array, i, min);
}

int main() {
    int a[5] = {5, 4, 3, 5, 6};
    int max = 10;
    int i = 0;

    cout << arrayMin(a, i, max) << endl;
    system("pause");
    return 0;
}
```



```
4
sh: pause: command not found
Program ended with exit code: 0
```

Рисунок 9.1 — Розв'язок задачі у ET

У заздалегідь підготовленому масиві найменшим елементом було число "4 завдяки чому можна стверджувати, що обчислення виконано правильно, так як отримані значення співпали.

### **9.5 Програми та обладнання.**

OpenOffice Calc, OpenOffice Draw, Xcode

### **9.6 Висновки.**

Під час виконання цієї практичної роботи були здобуті практичні навички зі створення та використання рекурсивних функцій